遺伝的アルゴリズムとSRGM に基づくオープンソースソフトウェアの最適バージョンアップ時期の推定に関する一考察(モデリングと最適化の理論)

AUTHOR(S):

田村, 慶信; 山田, 茂

CITATION:

田村, 慶信 ...[et al]. 遺伝的アルゴリズムとSRGM に基づくオープンソースソフトウェアの最適バージョンアップ時期の推定に関する一考察(モデリングと最適化の理論). 数理解析研究所講究録 2006, 1526: 11-19

ISSUE DATE:

2006-12

URL:

http://hdl.handle.net/2433/58883

数理解析研究所講究録
1526巻 2006年 11-19

遺伝的アルゴリズムとSRGMに基づく オープンソースソフトウェアの 最適バージョンアップ時期の推定に関する一考察

田村 慶信

広島工業大学情報学部
情報工学科
〒 731-5193
広島市佐伯区三宅 2-1-1
Phone: 082-921-6042
Fax: 082-921-8978
E-mail: tam@cc.it-hiroshima.ac.jp

山田 茂

鳥取大学工学部
社会開発システム工学科
〒 680-8552
鳥取市湖山町南 4-101
Phone: 0857-31-5303
Fax: 0857-31-0882
E-mail: yamada@sse.tottori-u.ac.jp

概 要

近年は，インターネットの普及により世界中が同時に新しい情報を得ることができるようになっている．こうした状況から，ネットワーク技術を基にしたソフトウェア製品の分散開発，およびソフトウェアそのものの分散化がさらに拡大してきた．中でも，オープンソースプロジェクトの下で開発されたオープンソースソフトウェアは，世界中の誰もが開発に参加でき，ソースコードが公開され，誰でも自由に改変することが可能なソフトウェアであることから，この数年急激に普及してきている．

本論文では，オープンソースソフトウェアのシステム全体を構成する複数のコンポーネントの信頼性に関する重要度を推定するために，遺伝的アルゴリズムを適用した信頼性評価法を提案する．同時に，ソフトウェア信頼度成長モデルに基づき各コンポーネント間の相互作用を包括した信頼性評価法を提案する．さらに，本手法に基づく最適なバージョンアップ時期の推定法について考察する．

1 はじめに

現在，分散ソフトウェア共同開発は，同一企業内における開発形態から，複数のソフトウェアハウスや同一企業内，複数の企業間での遠隔地間共同開発，さらには，多くの開発者が協調しながら開発を行うオープンソースプロジェクトなどの様々な形態が存在する．これに伴い，最近のソフトウェア開発は，クライアント/サーバ・システムやWebプログラミング，オブジェクト指向開発，ネットワーク環境での分散開発といった新しい開発技術が多用されるようになってきている [1, 2, 3]．特に，ネットワーク環境を利用して開発されたオープンソースソフトウェア（Open Source Software，以下OSSと略す）は，世界中の誰もが開発に参加でき，ソースコードが公開され，誰でも自由に改変することが可能なソフトウェアであることから，組込みシステムやサーバ用途として広く採用されている．また，OSSの代表格ともいうべきLinux[1]は市場でサーバ用途として有効であると認められ，この数年急激に普及してきている [4]．

オープンソースプロジェクトは，世界中に分散した複数のコンポーネントが，何らかのプラットフォーム上で的確に機能しているとすれば，開発が迅速であるばかりか，競争原理によりある評価基準の下でベストなものが残っていくと考えられる．オープンソースプロジェクトが分散型開発モデルを採用して成功した例として，GNU/Linuxオペレーティングシステムや Apache ウェブサーバなど[2]が挙げられる [5]．

一方，OSSの利用に関しては，未だに多くの不安が残されている．まず第1に，システム導入後のサポートおよび品質上の問題といった利用者側の一般的な不安である．第2に，OSSは本当にビジネスになるのか，オープンソースのソフトウェアを事業化することによって自社製のソフトウェア商品までが市場を失うことにならないか，といった開発者側の不安である [6]．特に，サポートや品質上の問題については，OSSの普及を妨げる大きな要因として考えられている．本論文では，こうしたオープンソースプロジェクトの下で開発されているOSSに対して，

[1] Linux は，Linus Torvalds の米国およびその他の国における登録商標あるいは商標である．
[2] その他記載している会社名，商品名は一般に各社の商標または登録商標である．

テスト管理に関する問題の1つとして信頼性評価法について議論するとともに，実際に公開されているOSSに対する信頼性評価法の適用可能性と，その有効性について考察する．

特に，システム全体を構成する複数のコンポーネント，いわゆる各ソフトウェアコンポーネントの重要度を推定するために遺伝的アルゴリズムを適用する．これにより，バグトラッキングシステム上から得られたデータに基づき，各コンポーネント間の相互作用を包括した信頼性評価法として利用できるものと考える．同時に，ソフトウェア信頼度成長モデル（software reliability growth model，以下SRGMと略す）[7]に基づき各コンポーネント間の相互作用を包括した信頼性評価法を提案する．また，実際のフォールト発見数データに対する数値例を示す．さらに，OSSの最適なバージョンアップ時期の推定法についても考察する．

2 各コンポーネント間の相互作用

2.1 システム全体の信頼性に対する影響度

ソフトウェアの信頼性評価手法の開発において，各コンポーネントでのデバッグの状況やその良し悪しがシステム全体の信頼性に与える影響を考慮しようとする場合，プログラムパス，コンポーネントの規模，フォールト報告者のスキルなどの，様々に絡み合った要因を捉える必要があると考えられる．これまでに，こうした複雑な状況下でシステム全体の信頼性に対する各コンポーネントの影響度合いを推定するために，主観的判断に基づく意思決定手法のAHP（Analytic Hierarchy Process）を利用した信頼性評価法が提案されている[8, 9]．AHPを利用した信頼性評価法は，各コンポーネントのバグフィックスの状態や実際の使用環境における各コンポーネント間でやりとりされるデータ量といったような開発リーダしか知り得ない情報を含んだ信頼性評価法として有用である．

しかしながら，オープンソースプロジェクトは，開発リーダが世界中に分散しているため，リーダ的人物は存在しているものの，実質上の開発リーダを特定しづらいことから，AHPを適用する上において，最も重要である評価基準の値を決定することが困難となる．また，AHPは評価基準の値に大きく左右されるため，代入された評価基準の値に誤りがあると信頼性評価結果自体も間違ったものとなることから，評価基準の決定に神経を使い，結果的には主観の決定に時間と労力を注ぎ込んでしまうという問題点があった．

したがって，本論文ではOSSのシステム構成について熟知していないユーザの立場に立った信頼性評価法として，遺伝的アルゴリズムを適用する．これにより，開発リーダの主観を必要とせず，バグトラッキングシステム上の採取されたデータのみに基づいて機械的に各コンポーネントのシステム全体に与える重要度を推定することが可能となる．

2.2 遺伝的アルゴリズムに基づく信頼性評価

本論文では，OSSのシステム全体の信頼性に対する各コンポーネントの影響度合い，つまり重要度を推定するために，遺伝的アルゴリズムを適用した信頼性評価法を提案する．これにより，各コンポーネント間の内部状態をブラックボックスとして捉えることにより，各コンポーネント間の相互作用の状態に対して物理的な意味合いを持たせることなくパラメータを推定することが可能となる．また，システム全体を構成するコンポーネント数が多い場合においても，各コンポーネントがシステム全体の信頼性に与える影響度合いを推定することが可能となる．これにより，バグトラッキングシステム上から採取されたデータのみに基づいた信頼性を評価することができるため，実利用上においても容易に適用できるものと考える．

2.3 遺伝的アルゴリズム

遺伝的アルゴリズム（Genetic Algorithm，以下GAと略す）は，生物の遺伝と進化のメカニズムを工学的にモデル化して，さまざまな問題解法やシステムの学習などに応用しようとするものである[11]．コンピュータ上に仮想生命を生成し，その環境に対する適応度を最適化問題の目的関数に一致させ，進化の過程をシミュレーションすることで，最適化問題を解くことが可能となる．

GAでは各個体を染色体によって特徴づける．染色体は複数個の遺伝子の集まりで構成される．生物は，特定の個数の染色体の集まりによって個体が決定されるが，GAでは1つの染色体で個体を表現することが多い．また，複数の個体間の相互協力によって解を探索するため，実用時間内に比較的優れた解を求めることができ，局所解に陥る可能性が低いという利点がある．さらに，突然変異という特徴により，OSSの各コンポーネントの重要度推定の際においてもコンポーネント数が多い場合でも適用できるという利点がある．システム全体に対する各コンポーネントの重要度 p_i の推定方法を以下に示す．

Step. 1

初期個体（染色体）をランダムに生成し，初期個体集合を生成する．その後，評価値の計算を行う．評価値は，初期個体集合を2進数にビット変換することにより表される．ビット列にコーディングを行うことで，各遺伝子がそのビット固有の意味を表し，他の遺伝子と相互作用がないため極めて応用範囲の広いものであるといえる [12].

Step. 2

第 a 世代の全ての染色体より，任意の2つの親となる個体をランダムに選び，選択された個体間の染色体の組み換えにより新しい個体を生成するための交叉を行う．これは，GA では最も重要な遺伝的オペレータであるといえる．この交叉は，選択された2つの個体をどのように交叉させるか，生成された新しい個体をいかにして個体群の中に組み込むのかという操作が重要になる [12].

本論文では，交叉方法として1点交叉を適用する．この1点交叉は単純交叉とよばれる最も単純な交叉規則である．具体的には，親の文字列上で交叉点を1ヵ所選び，交叉点より右側の2つの親の部分文字列をそのまま交換して，新しく子を2つ生成する．

Step. 3

各個体の評価値から適応度を計算する．適応度は評価値と，バグトラッキングシステム上の採取されたデータの評価関数から算出される．評価関数は以下の式で表される．

$$f_i = \sum_{i=1}^{n} a_i \cdot x_i. \tag{1}$$

ここで，a_i は評価対象となったフォールトの種類を表す．本論文では，a_1 は致命的であると判断されたフォールト数を各コンポーネントに対して正規化した値，a_2 は特定の OS において発見されたフォールト数を各コンポーネントに対して正規化した値，a_3 はシステムの内部構造に習熟した修正者のフォールト修正数を各コンポーネントに対して正規化した値，a_4 はシステムの内部構造に習熟した発見者のフォールト発見数を各コンポーネントに対して正規化した値，a_5 は各コンポーネントに対する累積フォールト発見数を各コンポーネントに対して正規化した値を取り上げた．また，x_i は $x_i = y_i / \sum_{i=1}^{n} y_i$ により定義する．ここで，y_i は a_i の各コンポーネントに対して正規化した値の平均値を表す．すなわち，x_i は a_i の重要度を表すものとする．

式 (1) の評価値と実際のバグトラッキングシステム上から採取されたデータの評価関数との差を，適合度として推定する遺伝的アルゴリズムを考える．

$$\begin{aligned} &\min_{x} \ F_i(x) \\ &F_i(x) = f_i - \frac{x}{2^n - 1}. \end{aligned} \tag{2}$$

ここで，x は染色体を2進数とみなした値であり生成された擬似乱数を表す．また，n は遺伝子長を表す．

Step. 4

ここまでの過程により，個体数は $(N + 2 \times a)$ 個となっている．N は初期個体集団，a は世代数を表す．ここで，環境に適合しない個体を淘汰することによって，個体数を N に戻す．方法としては，個体中で最も適応度の高い個体はそのまま次世代に残すという，エリート保存方式を適用する．この理由として，エリート保存方式以外の場合には，後に出てくる交叉と突然変異を行う際に，非常に良い個体が現れてもすぐに消滅してしまうことがある．これは確率的な操作をする以上やむを得ないことであり，局所解に陥ることを避けることにもつながるのであるが，現実に少ない回数で解を得たい場合に有効であることが挙げられる．

Step. 5

交叉だけでは，個体の親に依存するような限られた範囲の子しか生成することができないため，与えられた確率で突然変異を行う．この突然変異は染色体上のある部分の値を強制的に対立遺伝子に置き換えることにより，交叉だけでは生成できない子を生成して，個体群の多様性を維持する働きをする．これにより，より良い解を持つ個体の発生を期待するのであるが，突然変異率をあまり大きくし過ぎると悪い方向への変異の確率も大きくなり，解を得るのが難しくなる [11].

本論文では突然変異率を 0.01 とし，各遺伝子をランダムに 1ヵ所対立遺伝子に置き換える．

Step. 6

世代が一定数に達するまで，Step. 2〜Step. 5 を繰り返す．

以上の操作により，各コンポーネントに対する重要度 $p_i(i = 1, 2, \cdots, n)$ が算出される．本論文では，遺伝的アルゴリズムにおいて推定された各コンポーネントの重要度 p_i を，システム全体の信頼性に対する各コンポーネントの影響度合いと定義する．これは，あくまで信頼性に関する重要度であって，各コンポーネントの開発規模などを表すものではない．

3 システム全体に対する信頼性評価

3.1 一般化対数型ポアソン実行時間モデル

本論文で取り上げる OSS の動作環境は，様々なアプリケーションソフトウェアから影響を受け易く，従来のような同一組織内で開発され，単体で動作するソフトウェアシステムとは大きく環境が異なる．こうしたソフトウェア間の相互作用により，発見されるフォールト数も一定の値に収束することなく，将来的には増加し続けるものと考えられる．

本論文では，検出可能フォールト数が無限であると仮定された NHPP に基づく対数型ポアソン実行時間モデルを適用する．時間区間 $(0, t]$ で発見される総期待フォールト数を表す平均値関数 $\mu(t)$ は，

$$\mu(t) = \frac{1}{\theta - P} \ln[\lambda_0(\theta - P)t + 1] \quad (0 < \theta,\ 0 < \lambda_0,\ 0 < P < 1), \tag{3}$$

により与えられる．ここで，パラメータ λ_0 は初期故障強度，パラメータ θ はソフトウェア故障 1 個当りの故障強度の減少率を表す．また，パラメータ P はシステム全体に及ぼすコンポーネントの影響率を表す．これは，各コンポーネントに対して遺伝的アルゴリズムを用いて推定されたパラメータ y_i の平均値により表されるものと定義する [8, 9, 10, 13]．

3.2 ソフトウェア信頼性評価尺度

式 (3) の平均値関数をもつ NHPP モデルから，種々のソフトウェア信頼性評価のための定量的尺度を導出できる．

瞬間フォールト発見率は強度関数により表すことができる．これは，単位時間当りに発見されるフォールト数として定義される．瞬間フォールト発見率は，式 (3) から以下のように導出できる．

$$\mu_d(t) = \frac{d\mu(t)}{dt}. \tag{4}$$

平均ソフトウェア故障発生時間間隔（mean time between software failures: MTBF）は，ソフトウェア 故障の発生頻度を表すのに有益な尺度である．また，MTBF が大きな値を取ることは，それだけフォールトが発見し難くなり，ソフトウェア信頼性が向上したと判断できることになる．任意の時刻 t における瞬間 MTBF（instantaneous MTBF: $\mathrm{MTBF_I}$）および累積 MTBF（cumulative MTBF: $\mathrm{MTBF_C}$）は，以下のように導出できる．

任意の時刻 t における瞬間的なフォールト発見間隔の平均を意味する瞬間 MTBF は，

$$MTBF_I(t) = \frac{1}{d\mu(t)/dt}, \tag{5}$$

となる．

運用開始時点から考えたときの発見フォールト 1 個当りに要する発見時間の平均を意味する累積 MTBF は，

$$MTBF_C(t) = \frac{t}{\mu(t)}, \tag{6}$$

により表すことができる．

4 実測データに適用した数値例

4.1 各コンポーネントに対する信頼性評価

実際のオープンソースプロジェクトにおけるバグトラッキングシステムから採取されたフォールトデータを適用した数値例を示す．本論文で提案された信頼性評価法の性能評価を行うために，Mozilla プロジェクト[3]で開発

[3]Mozilla と Mozilla のロゴは Mozilla Foundation の登録商標である．

表 1： Thunderbird の各コンポーネントに対する要因別データ.

Component Name	Number of Fault Level (Critical)	OS (Windows)	Assigned to (mscott)	Reporter (floeff)	Total Number of Faults
Account Manager	9	88	127	1	130
Address Book	4	59	102	3	105
Build Config	1	3	14	0	15
General	72	293	538	3	558
Help Documentation	0	1	3	0	5
Installer	3	28	33	1	33
Mail Window Front End	58	578	982	6	1013
Message Compose Window	15	127	227	6	233
Migration	3	36	51	0	51
Preferences	2	37	92	5	96
RSS	3	45	76	0	78
Total	170	1295	2245	25	2317

表 2： Thunderbird の各コンポーネントに対する重み係数.

Component Name	Weight parameter
Account Manager	0.0537
Address Book	0.0487
Build Config	0.0091
General	0.2377
Help Documentation	0.0054
Installer	0.0138
Mail Window Front End	0.4347
Message Compose Window	0.0951
Migration	0.0266
Preferences	0.0425
RSS	0.0327

が進められているメーラの Thunderbird[4][14] と呼ばれる OSS を取り上げる.

各コンポーネントに関して，システム全体の信頼性に影響を及ぼすと考えられる要因別データ一覧を表 1 に示す. 遺伝的アルゴリズムを適用するにあたり，表 1 におけるデータを入力データとした. **2.2** の遺伝的アルゴリズムに基づく各 OSS の各コンポーネントに対する重みパラメータ $p_i(i = 1, 2, \cdots, n)$ の推定結果を表 2 に示す. 表 2 から，Mail Window Front End コンポーネントに対する重要度が最大であることが分かる. 一方，Help Documentation コンポーネントに対する重要度は最小であることが確認できる.

4.2 システム全体に対する信頼性評価

次に，システム全体に対する信頼性評価結果の一例を示す. まず，式 (3) における累積フォールト発見数の期待値の推定値 $\widehat{\mu}(t)$ を図 1 に示す. さらに，式 (3) の平均値関数から導出される瞬間フォールト発見率の推定値 $\widehat{\mu_d}(t)$，瞬間 MTBF の推定値 $\widehat{MTBF_I}(t)$ および累積 MTBF の推定値 $\widehat{MTBF_C}(t)$ の推定結果を図 2 から図 4 に示す. これらの結果から，瞬間 MTBF および累積 MTBF は両者ともに，時間の経過とともに平均故障発生時間間隔が小さくなり，今後もソフトウェア故障が頻繁に発生することが確認できる.

5 最適バージョンアップ

5.1 最適バージョンアップ時期の推定

OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定することは，リリース後の信頼性維持や進捗度管理に役立つと考えられる. 本論文では，オープンソースプロジェクトの下で開発された Thunderbird[14] を一例に挙げ，OSS のバージョンアップ時期の推定方法について議論する.

[4]Thunderbird と Thunderbird のロゴは Mozilla Foundation の米国及びその他の国における商標または登録商標である.

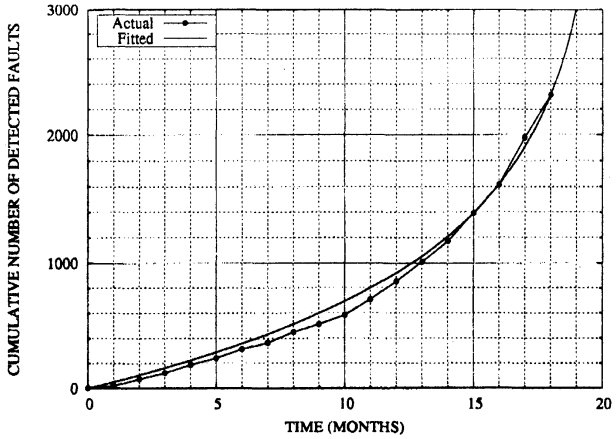

図 1： 推定された累積フォールト発見数の期待値，$\widehat{\mu}(t)$.

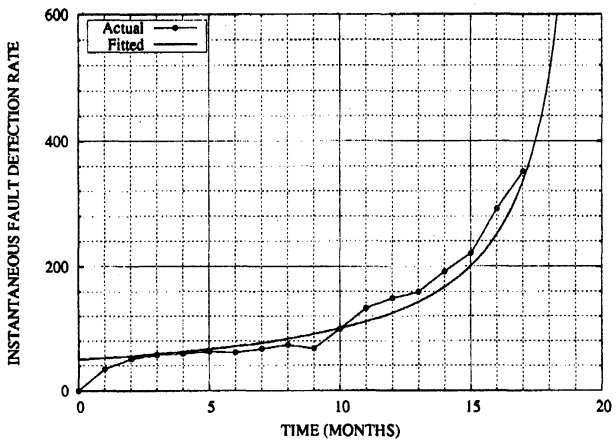

図 2： 推定された瞬間フォールト発見率，$\widehat{\mu_d}(t)$.

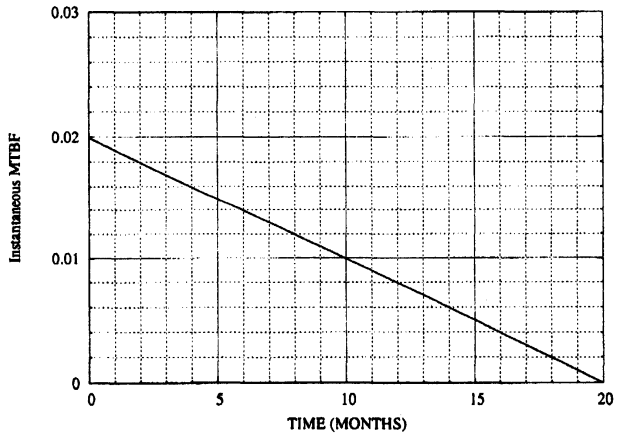

図 3： 推定された瞬間 MTBF，$\widehat{MTBF_I}(t)$.

バージョンアップには大きく分けてマイナーバージョンアップとメジャーバージョンアップがある．しかしながら，どの程度の改訂で区別されるという明確な基準がある訳ではない．マイナーバージョンアップは，旧版にいくつかの細かい機能が追加されたり，性能が若干向上した場合に実施される．ただし，機能の不具合やセキュリティ上の脆弱性を修復するような，クリティカルなフォールトがいくつか発見され緊急に修正を施す必要があ

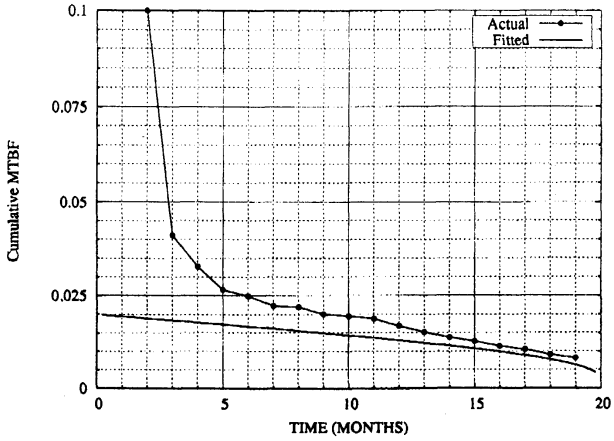

図 4： 推定された累積 MTBF，$\widehat{MTBF_C}(t)$.

る場合に実施される改訂はマイナーバージョンアップではなく，バグフィックスと呼ばれている．一方，メジャーバージョンアップは，OSS 自体の機能が大きく変更されたり，大型の新機能の追加や，性能が劇的に向上した場合に実施される．OSS におけるマイナーバージョンアップは，不定期かつ頻繁に実施されていることから，その推定時期を予測することは困難であり，たとえマイナーバージョンアップ時期が推定できたとしても，その有益性は小さいと考えられる．したがって，本論文では，メジャーバージョンアップを対象とし，前回のバージョンアップ期間に基づいて，次期バージョンアップ時期を予測するための手法について考察する．

5.2 総開発労力の定式化

OSS の開発に伴う総労力を定式化し，総開発労力を最小にする時刻を最適バージョンアップ時刻と定義することにより，バージョンアップ後の信頼性維持や進捗度管理に役立つものと考える．まず，総開発労力を定式化するために，以下のパラメータを定義する．

$m_{1,i}$: コンポーネント i のフォールト修正に伴う修正労力 $(m_{1,i} > 0)$,

$m_{2,i}$: コンポーネント i の単位時間当りの開発労力 $(m_{2,i} > 0)$,

m_3: メジャーバージョンアップ後のフォールト修正に伴う保守労力 $(m_3 > 0)$,

m_4: メジャーバージョンアップ後の単位時間当りの保守労力 $(m_4 > 0)$.

よって，以下のような各コンポーネントに対する期待開発労力が得られる．

$$E_1(t_i) = \sum_{i=1}^{n} \{m_{1,i} \cdot p_i \cdot \mu(t_i) + m_{2,i} \cdot t_i\} \qquad (i = 1, 2, \cdots, n). \tag{7}$$

ここで，t_i は i 番目のコンポーネントに対する開発期間を表す．

一方，メジャーバージョンアップ後の保守労力は以下のように定式化できる．

$$E_2(t) = m_3\{\mu(t_0) - \mu(t)\} + m_4 t. \tag{8}$$

ここで，t_0 は過去のバージョンアップ期間の平均値を表す．

さらに，時間区間 $(0, t_0]$ を越えた場合において，各コンポーネントを新規に開発する際には，整合性を確認するためのペナルティ労力が課せられるものとする．本論文では，ペナルティ関数を以下のように定義する．

$$G_i(t) = \begin{cases} c_{3i}\left\{e^{k_i(t-t_{di})} - 1\right\} & (t > t_{di}) \\ 0 & (t \leq t_{di}). \end{cases} \tag{9}$$

ここで，t_{di} はコンポーネント i の前回のバージョンアップ時刻 $(t_{di} > 0)$，$c_{3i}(> 0)$ および $k_i(> 0)$ は定数パラメータを表す．

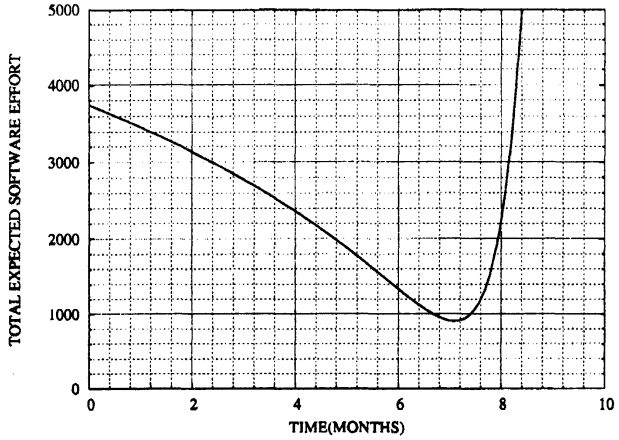

図 5： 推定された総期待開発労力.

したがって，総期待開発労力は，式 (7)，式 (8) および式 (9) より，

$$E(t) = E_1(t) + E_2(t) + \sum_{i=1}^{m} G_i(t) \tag{10}$$

のように表すことができる．ここで，m はメジャーバージョンップ後において新規に開発されたコンポーネント数を表す．この式 (10) を最小にする時刻 t^* が，OSS の最適メジャーバージョンアップ時刻となる．

5.3 数値例

最適メジャーバージョンアップ時期推定の数値例として，Thunderbird を取り上げる．ここでは一例として，Ver. 1.6 から Ver. 1.7 への移行時期に基づき，Ver. 1.8 への時期バージョンアップ時期を推定する．Ver. 1.6 は 2004 年 1 月 15 日にリリースされ，Ver. 1.7 は 2004 年 6 月 17 日にリリースされており，この間の期間は 111 日である．したがって，t_0 を 111 と仮定し，2004 年 6 月 17 日以降における式 (10) の推定された総期待開発労力を図 5 に示す．図 5 から，最適バージョンアップ時刻は Ver. 1.7 がリリースされてから約 7.09ヶ月目となり，そのときの総期待開発労力は 907.02 であることが確認できる．

6 おわりに

本論文では，オープンソースプロジェクトの下で分散共同開発されている OSS に対する信頼性評価法について議論した．特に，ユーザ指向の信頼性評価法として遺伝的アルゴリズムを適用することにより，各コンポーネントに対する相互作用を包括した信頼性評価法について議論した．また，実際の OSS のバグトラッキングシステムから採取されたフォールト発見数データに対する数値例を示した．

本手法により，システム全体の信頼性に与える各コンポーネントの重要度を定量的に導出することが可能となる．さらに，導出された各コンポーネントの重み係数から各コンポーネントに対する発見フォールト数を推定することが可能となることから，OSS の信頼性を定量的に把握するための手段として有効であると考える．さらに，OSS の開発において，ある程度目安となるような適切なバージョンアップ時期を推定するために，最適バージョンアップ時期の推定方法について議論した．本論文において提案された手法によって，OSS のバージョンアップ後の信頼性維持や進捗度管理に役立つと考えられる．特に，最適バージョンアップ時期の推定のために，OSS の総期待開発労力を定式化した．しかしながら，フォールト修正に伴う修正労力や単位時間当りの開発労力に関するパラメータの決定方法が曖昧であることから，今後はこれらのパラメータの厳密な設定方法について議論する必要がある．

オープンソースという開発スタイルは，今後も何らかの形で市場で大きな流れを作っていくものと考えられる．この流れを阻害する大きな要因として，サポートや品質上の問題が挙げられる．本論文では，こうした問題を解決するために，オープンソースプロジェクトの下で開発された OSS に対する信頼性評価法の 1 例を示した．

将来的には，本手法をソフトウェアツールとして実装することにより，多くのユーザに対して容易に扱うことが可能な信頼性評価ツールとして提供していくことが重要であると考える．これまで，OSS では信頼性を定量的に評価するという試みが行われていなかったことから，本論文において新たに提案された信頼性評価手法を OSS に適用することによって，より高品質な OSS の開発に結びつくものと考える．

謝辞

本論文の一部は，文部科学省科学研究費基盤研究 (C)（課題番号 18510124）および若手研究 (B)（課題番号 17700039）の援助を受けたことを付記する．

参考文献

[1] A. Umar, *Distributed Computing and Client-Server Systems*, Prentice Hall, New Jersey, 1993.

[2] 松本 正雄, 小山田 正史, 松尾谷 徹, ソフトウェア開発検証技法, 電子情報通信学会, 東京, 1997.

[3] 赤羽 豊和, クライアント/サーバ・システムのテスト技法, ソフト・リサーチ・センター, 東京, 1998.

[4] トロン協会 ITRON 仕様検討グループ, ITRON Project Archive, http://www.sakamura-lab.org/TRON/ITRON/home-j.html

[5] E-Soft Inc., Internet Research Reports, http://www.securityspace.com/s_survey/data/index.html

[6] ソフトウェア情報センター研究会報告書, オープンソースソフトウェアの利用状況調査／導入検討ガイドラインの公表について, 東京, 2004.

[7] 山田 茂, ソフトウェア信頼性モデル－基礎と応用－, 日科技連出版社, 東京, 1994.

[8] 田村慶信, 山田茂, 木村光宏, “オープンソース共同開発環境に対するソフトウェア信頼性評価法に関する考察,” 電子情報通信学会論文誌, vol. J88-A, no. 7, pp. 840–847, 2005.

[9] Y. Tamura and S. Yamada, “Comparison of software reliability assessment methods for open source software,” *Proceedings of the 11th IEEE International Conference on Parallel and Distributed Systems (ICPADS2005)-Volume II*, Fukuoka, Japan, July 20–22, pp. 488–492, 2005.

[10] Y. Tamura and S. Yamada, “A Method of User-oriented Reliability Assessment for Open Source Software and Its Applications,” *Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics*, Taipei, Taiwan, Oct. 8–11, 2006, pp. 2185–2190.

[11] 萩原将文，ニューロ・ファジィ・遺伝的アルゴリズム，産業図書，東京，1994.

[12] 坂和正敏，田中雅博，遺伝的アルゴリズム，朝倉書店，東京，1995.

[13] 田村慶信，肌附康司，山田茂，木村光宏，“オープンソースソフトウェアに対するユーザ指向の信頼性評価ツールの開発，” Linux Conference，東京，2006，http://lc.linux.or.jp/lc2006/

[14] The Mozilla Thunderbird Mail Project, Thunderbird, http://www.mozilla.org/projects/thunderbird/